

## APPENDIX (SOURCE CODE)

/\*-----

Text Rain Source Code

written by Camille Utterback

-----\*/

#include <windows.h>

#include <stdlib.h>

#include <string.h>

#include <vfw.h>

#include <time.h>

#include <wingdi.h>

#define MAXNUMLINES 25

//the max number of lines that can be entered at one time (this is just a practical limmit for testing  
- no real reason to keep it this low

#define MAXNUMLETTERS 60

//this could be flexible based on the screen resolution

#define MAXREAD 8192

//biggest amt of text to read into our buffer

#define PINK RGB(200,100,155)

#define GOLD RGB(185,140,65)

#define BLUE RGB(25,0,190)

#define RED RGB(210,0,30)

#define GREEN RGB(0,210,0)

//--DROP struct - one for each letter

typedef struct

{

char ltr;

int x;

int y;

int rate;

COLORREF fadeColor;

}

DROP;

//--LINE struct - an array of drops, there will be one for each line

typedef struct

{

DROP dropline[MAXNUMLETTERS]; //should call this dropArray

//int dropArrayLength OR LINEINFO LineInfo

}

LINE;

//--LINEINFO struct - info about each line (should really be part of LINE)

typedef struct

{

bool move;

bool fade;

COLORREF color;

//int length;

}

LINEINFO;

/\* -----Declare all functions -----\*/

LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM); //windows base routine

void Run(HWND hwnd); //prints everything from the buffer to the screen

void CleanUp(); //closes stuff before quitting

int FindDriver(char driverstring[80], int stringlength);

bool ReadInTextFromFile(LPCTSTR szFileName,LPVOID szBuffer,DWORD maxcharstoread,LPDWORD lpcharsread);

int SetArray(char[], COLORREF, DROP[]); //returns length of array

LRESULT CALLBACK VideoCallbackProc(HWND, LPVIDEOHDR); //video callback function

```
void SetToTop(COLORREF,DROP[],int);
```

```
void IncLocAndRgn(DROP[], int);
```

```
void FadeText(DROP[], int);
```

```
void PrintText(DROP[], int);
```

```
bool CheckPixel(int, int);
```

```
bool CheckLevel(int);
```

```
void AperatureTest();           //for ceiling projection
```

```
void AperatureFix();
```

```
void DrawBlackRect(HDC whichdc, int left, int top, int right, int bottom);
```

```
/*-----Declare all global variables -----*/
```

```
//-----STRING
```

```
LINE Lines[MAXNUMLINES]; //an array of LINE structs (which are each an array of DROP  
structs
```

```
LINEINFO LineInfos[MAXNUMLINES];
```

```
int LineLengths[MAXNUMLINES]; //array that will hold the letter length of each line in the  
poem
```

```
//could incorporate this into lineinfos.
```

```
COLORREF colors[]={GREEN,PINK,GOLD,BLUE,RED};
```

```
int NumColors;
```

```
int gNumLines;
```

```
//-----GENERAL
```

```
int cxClient, cyClient; // window dimensions
```

```
TEXTMETRIC tm;
```

```
int cxChar, cyChar, cxCaps; //x and y dimensions of text
```

```
char buffer[MAXREAD];
```

```
int CursorValue=0;
```

```
int ActiveLineIndex=0;
```

```
int HoleWidth=4;
```

```
int HoleHeight=4;
```

```
int HoleL=309;           //int HoleL=(640)-(HoleWidth/2);
```

```
int HoleT=237;          //int HoleT=(480)-(HoleHeight/2);
```

```
int HoleGrey=0;
```

```
int DarkThresh=180;
```

```
bool firstpaint=1;
```

```
//-----WINDOW/OBJECTS
```

HWND hwnd; //main window

HWND ghWndCap; //capture window

HDC hdc;

HDC viddc; //memdc for video image

HBITMAP tempbitmap2;

HDC backwardsdc; //memdc for flipping video image

HBITMAP BackwardsBitmap;

HDRAWDIB hdd; //to decompress captured info

//-----VIDEO

HGLOBAL hgout;

DWORD dwsize;

LPVOID pformatbmiv;

LPBITMAPINFO pformatbmi;

unsigned char\* pbits;



```
/*-----*/
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst, PSTR szCmdLine, int  
iCmdShow)
```

```
{
```

```
    static char szAppName[] = "Text Rain";
```

```
    MSG msg;
```

```
    WNDCLASSEX wndclass;
```

```
    int i;
```

```
    /*define a window class*/
```

```
    wndclass.cbSize = sizeof(wndclass);
```

```
    wndclass.style = 0; /*0 was the default style*/
```

```
    wndclass.hInstance = hInstance; /*handle to this instance*/
```

```
    wndclass.lpszClassName = szAppName; /*window class name*/
```

```
    wndclass.lpfnWndProc = WindowProc; /*window function*/
```

```
    wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); /*icon style*/
```

```
wndclass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

wndclass.hCursor = LoadCursor(NULL, IDC_ARROW); /*cursor style*/

wndclass.lpszMenuName = NULL; /*no menu*/


wndclass.cbClsExtra = 0; /*no extra*/

wndclass.cbWndExtra = 0; /*info needed*/


wndclass.hbrBackground =(HBRUSH) GetStockObject(WHITE_BRUSH);


RegisterClassEx (&wndclass);


hwnd = CreateWindow(

    szAppName, /*name of window class*/

    NULL, /*title */

    WS_POPUP, /*window style*/

    0, /*x coordinate*/

    0, /*y coordinate*/

    640, /*width coordinate*/

    480, /*height coordinate*/

    NULL, /* or HWND_DESKTOP no parent window*/

    NULL, /*menu handle (if parent) - child id if window is a child*/
```

```

        hInstance, /*handle of this instance of the program*/

        NULL /*no additional arguments*/

    );

    /*windows sends WM_CREATE msg to wndProc while processing createwindow */

    /*display window*/

    ShowWindow(hwnd, iCmdShow); /*windows sends WM_SIZE,
WM_SHOWWINDOW*/

    UpdateWindow(hwnd); /* sends WM_PAINT message to windProc */

    //-----

    while (TRUE)
    {

        if (PeekMessage (&msg, NULL, 0,0, PM_REMOVE))

        {

            if (msg.message == WM_QUIT)

                break;

            TranslateMessage(&msg);

            DispatchMessage(&msg); /* sends msg to windows which calls wndProc
with it*/

```

```

char szTextFile[]="words.txt";      //name of file to read from

static char sReadBuffer[MAXREAD];    //our buffer to read stuff into

static char szTempBuffer[MAXREAD];

DWORD charsreadok;


int i,j,linenum,t,g;

int MaxLineLength;

int ColorIndex;


switch(iMsg)
{

case WM_CREATE:

    //-----get device context

    hdc=GetDC(hwnd);


    //-----create VIDDC

    viddc=CreateCompatibleDC(hdc);

    tempbitmap2=CreateCompatibleBitmap(hdc, 640, 480);

    SelectObject(viddc,tempbitmap2); //do I even need to do this?

```

```
SetBkMode(viddc, TRANSPARENT);
```

```
//-----CREATE BACKWARDSDC
```

```
backwardsdc=CreateCompatibleDC(hdc);
```

```
BackwardsBitmap=CreateCompatibleBitmap(hdc, 640, 480);
```

```
SelectObject(backwardsdc,BackwardsBitmap); //enlarge dc from 1 monochrome
```

pixel

```
//-----CREATE HDRAWDIBs
```

```
hdd=DrawDibOpen();
```

```
//-----get Text Metrics
```

```
GetTextMetrics(hdc,&tm);
```

```
cxChar=tm.tmAveCharWidth+1;
```

```
cyChar=tm.tmHeight+tm.tmExternalLeading+1;
```

```
cxCaps= (tm.tmAveCharWidth * 3)/2+1; //cxChar=avecharwidth
```

```
//-----READ IN USER'S ascii STRING and SET ARRAY
```

```
if(!((ReadInTextFromFile(szTextFile,sReadBuffer,MAXREAD,&charsreadok))
&& (strlen(sReadBuffer)>1))) {
```

```
    //---readtext didn't work or string is only one char long
```

```
    MessageBox( hwnd, // handle of owner window
                "can't read in text", // address of text
                in message box
                NULL, // address of title of message
                box
                MB_OK // style of message box
            );
```

```
    CleanUp();
    PostQuitMessage(0); //user clicked close, puts WM_QUIT
    message in the message queue
```

```
    } else {
```

```
        //---readtext in from file OK, parse through it.
```

```
        MaxLineLength=MAXNUMLETTERS; //length of longest line in textrain
        arrays.
```

```
        NumColors=(sizeof(colors))/(sizeof(COLORREF));//5
```

```
        ColorIndex=0;
```

```
        linenum=0;
```

```
        i=0;
```

```
        j=0;
```

```
while(i<charsreadok){ //strlen(buffer)-1?
```

```
if((j<MaxLineLength)&&(sReadBuffer[i]!='\r')){
```

```
//---- need to add something to our temp array
```

```
if(sReadBuffer[i]=='\t'){
```

```
//---tab char - put 5 spaces into temp array
```

```
for(t=0;t<5;t++){
```

```
if(j<MaxLineLength){ //and eof?
```

```
szTempBuffer[j]=' ';
```

```
j++; //don't increment i here - it
```

```
}
```

```
}
```

```
}else{
```

```
//---all other chars
```

```
if(sReadBuffer[i]!='\n'){
```

```
//--not a newline, add to array
```

```
szTempBuffer[j]=sReadBuffer[i];
```

happens i

```
j++;
```

```
}
```

```
}
```

```
i++; //tab, newline, or normal char
```

```
} else { //end if(j<MaxLineLength)&&(sReadBuffer[i]!='r')
```

```
    //j> maxlenlength or we're at a cr
```

```
    //in either case - we're at the end of the line
```

```
    if(j>=MaxLineLength){ //move I to next cr
```

```
        while((sReadBuffer[i] != '\r') && (i<(charsreadok)))
```

```
{
```

```
            i++;
```

```
}
```

```
}
```

```
i++; //i= cr whether or not we hit maxline, move to next
```

letter

```
//----SET ARRAY
```



times) //send temparray to SetArray (or do this numrepeatlines

//--- set up LineInfos struct for this line

if(linenum<2)

LineInfos[linenum].move=1;

else

LineInfos[linenum].move=0;

LineInfos[linenum].fade=0;

LineInfos[linenum].color=colors[ColorIndex];//RGB(200,100,155)

if(ColorIndex<(NumColors-1))

ColorIndex++;

else

ColorIndex=0;

//--- send info to SetArray

LineLengths[linenum] = SetArray(szTempBuffer,  
LineInfos[linenum].color, Lines[linenum].dropline);

struct for each letter,  
(basically the strlen)

```
//this takes a string of text Text[i], fills a DROP  
//and returns the num of DROP structs filled
```

```
for(t=0;t<MaxLineLength;t++)  
    szTempBuffer[t]=' ';
```

```
j=0;  
linenum++;
```

```
    }  
}
```

gNumLines=linenum; //this will be the actual NUMBER of lines, not the  
last index num.

```
}//end else for reading in file
```

```
//-----CREATE CAPTURE WINDOW
```

```
ghWndCap = capCreateCaptureWindow((LPSTR)"Capture Window",  
WS_CHILD|WS_VISIBLE, 0,0,600,440,(HWND) hwnd, (int)0);
```

```
//-----register callback functions
```

```
capSetCallbackOnVideoStream(ghWndCap,&VideoCallbackProc);
```

```
return 0;
```

```
case WM_SIZE:
```

```
/*lParam that gets passed with this message contains  
the new width of the client window in the lowWord  
and the new height of the client window in the HiWord*/
```

```
cxClient=LOWORD(lParam);
```

```
cyClient=HIWORD(lParam);
```

```
//-----connect to the driver
```

```
    //if(capDriverConnect(ghWndCap, FindDriver(QuickCamString,  
(int)strlen(QuickCamString)))){
```

```
        if(capDriverConnect(ghWndCap, FindDriver(ATIString,  
(int)strlen(ATIString)))){
```

```
//-----get and set streaming vid parameters
```

```
capCaptureGetSetup(ghWndCap,&CapParms, sizeof(CapParms));
```

```
CapParms.fCaptureAudio=0; //no audio
```

```
CapParms.dwRequestMicroSecPerFrame=22222;
```

```
CapParms.fAbortRightMouse=0; //no action for rt mouseclick
```

```
CapParms.fAbortLeftMouse=0; //no action for lft mouseclick
```

```
capCaptureSetSetup(ghWndCap,&CapParms, sizeof(CapParms));
```

```
capPreviewScale(ghWndCap,1); //stretch preview to size of capture window
```

```
return 0;
```

```
}
```

```
else{
```

```
        MessageBox( hwnd, "can't connect to capture driver", NULL,  
MB_OK|MB_SYSTEMMODAL );
```

```
        CleanUp();
```

```
        //-----put WM_QUIT message in the message queue (your program won't  
get this back)
```

```
        PostQuitMessage(0);
```

```
        return 0;
```

```
    }
```

```
case WM_PAINT:
```

```
    if (firstpaint==1){
```

```
        BitBlt(viddc, 0, 0, cxClient, cyClient, hdc, 0, 0, SRCCOPY); //save a copy
```

```
        firstpaint=0;
```

```
    }
```

```
    return 0;
```

```
case WM_LBUTTONDOWN:
```

//----video SOURCE dialog - BRIGHTNESS,CONTRAST etc for qc

//if cursor is invisible

if(CursorValue<0) //will be -1 if invis

CursorValue=ShowCursor(1);//show cursor

capDriverGetCaps(ghWndCap, &CapDrvCaps, sizeof(CAPDRIVERCAPS));

if (CapDrvCaps.fHasDlgVideoSource)

capDlgVideoSource(ghWndCap);

return 0;

case WM\_RBUTTONDOWN:

//----video FORMAT dialog box - IMAGE SIZE and QUALITY for qc

//if cursor is invisible

if(CursorValue<0) //will be -1 if invis

```
CursorValue=ShowCursor(1);//show cursor
```

```
capDriverGetCaps(ghWndCap, &CapDrvCaps, sizeof(CAPDRIVERCAPS));
```

```
if (CapDrvCaps.fHasDlgVideoFormat)
```

```
capDlgVideoFormat(ghWndCap);
```

```
return 0;
```

```
case WM_KEYDOWN:
```

```
//No Keydown messages are processed while capturing -
```

```
switch(wParam)
```

```
{
```

```
case VK_UP:
```

```
BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);
```

```
HoleT-=1;
```

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",  
HoleGrey));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"gNumLines = %d",  
gNumLines));

break;

case VK\_DOWN:

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

HoleT+=1;

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",  
HoleGrey));

break;



```
case VK_LEFT:
```

```
    BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);
```

```
    HoleL-=1;
```

```
    AperatureTest();
```

```
    TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));
```

```
    TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",  
HoleGrey));
```

```
break;
```

```
case VK_RIGHT:
```

```
    BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);
```

```
    HoleL+=1;
```

```
    AperatureTest();
```

```
    TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));
```

```
    TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",  
HoleGrey));
```

```
break;
```

case VK\_F1:

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

HoleHeight+=1;

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer, "loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer, "grey = %d",  
HoleGrey));

break;

case VK\_F2:

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

HoleHeight-=1;

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer, "loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer, "grey = %d",  
HoleGrey));

break;

case VK\_F3:

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

HoleWidth+=1;

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",  
HoleGrey));

break;

case VK\_F4:

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

HoleWidth-=1;

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

```
HoleGrey));
    TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",
```

```
break;
```

```
case VK_F5:
```

```
    /*
```

```
    BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);
```

```
    HoleGrey-=10; //-=darker
```

```
    HoleGrey=(HoleGrey<0)? 0:HoleGrey; //if r>255 make it 255,
else let it be
```

```
    AperatureTest();
```

```
    TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));
```

```
    TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",
HoleGrey));
```

```
    */
```

```
    BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);
```

```
    DarkThresh-=5;
```

DarkThresh));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"thresh = %d",

break;

case VK\_F6:

/\*

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

HoleGrey+=10;

HoleGrey=(HoleGrey>255)? 255:HoleGrey; //if r>255 make it  
255, else let it be

AperatureTest();

TextOut(hdc, 30, 30, buffer, wsprintf(buffer,"loc(L,T) = %d, %d  
w=%d h=%d", HoleL, HoleT, HoleWidth, HoleHeight));

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"grey = %d",  
HoleGrey));

\*/

BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

DarkThresh+=5;

TextOut(hdc, 30, 70, buffer, wsprintf(buffer,"thresh = %d",

DarkThresh));

break;

case VK\_SPACE:

//-----Get video format info and begin capture

pformatbmi  
//---1st get pointer to VideoFormat BITMAPINFO struct

BITMAPINFO for buffer  
dwsz=capGetVideoFormatSize(ghWndCap); //retrieve size of

global memory  
hgout=GlobalAlloc(GHND,dwsz); //allocate that amount of

start  
pformatbmiv=GlobalLock(hgout); //locks object, returns handle to

pformatbmi=(LPBITMAPINFO)pformatbmiv; //convert this  
pointer from void to bitmapinfo struct pointer

//LPBITMAPINFO pformatbmi;

//LPVOID pformatbmiv;

capGetVideoFormat(ghWndCap, pformatbmi, dwsz);

//valid BITMAPINFO struct is now at address pformatbmi

GlobalUnlock(hgout);

//-----CAPTURE!!

PatBlt(hdc, 0, 0, cxClient, cyClient, PATCOPY); //paint screen  
white to cover text etc

//make capture window invisible

ShowWindow(ghWndCap,FALSE);

capPreviewScale(ghWndCap,0);//turn off scaling to capture  
window (may improve speed)

CursorValue=ShowCursor(0);//hide cursor

capCaptureSequenceNoFile(ghWndCap);//capture but don't save

//-----after capture ends (with escape key)-----

PatBlt(hdc, 0, 0, cxClient, cyClient, PATCOPY);

capPreviewScale( ghWndCap, 1);

ShowWindow(ghWndCap,TRUE);

CursorValue=ShowCursor(1);

```
break;
```

```
case VK_RETURN: //only works after stopping capture w escape
```

```
    CleanUp();
```

```
    PostQuitMessage(0); //user clicked close, puts WM_QUIT  
message in the message queue
```

```
break;
```

```
} //end switch
```

```
return 0;
```

```
case WM_DESTROY: /*terminate the program*/
```

```
    //WM_CLOSE when processed by defwinproc generates a WM_DESTROY msg.
```

```
    //this is the last msg WinProc will get
```

```
    CleanUp();
```



```
        PostQuitMessage(0); /*user clicked close, puts WM_QUIT message in the
message queue*/
```

```
        //this causes GetMessage to return 0
```

```
    return 0;
```

```
    } // end msg switch
```

```
    return DefWindowProc(hwnd, iMsg, wParam, lParam); //let windows handle if I haven't
```

```
}
```

```
//-----
```

```
int SetArray(char myLineText[], COLORREF myLineColor, DROP mLetterArray[])
```

```
{
    //a character array          //a colorref          //an array of DROP
    structs to fill
```

```
    //this function loops through a line of text and fills each DROP struct in a corresponding
```

```
    //array with the proper letter and color.
```

```
    //it also assigns a starting x,y loc and rate to each DROP struct
```

```
//spaces are omitted (ie not added to the DROP array)
```

```
//but the x loc of the next char is adjusted appropriately
```

```
//for readin - count line, figure out center and where line should start (left margin)
```

```
//for now - if line is too long, just omit rest.
```

```
int i;
```

```
int j=0;
```

```
int count=0;
```

```
int size = strlen(myLineText); //length of the string (character array) we're passing
```

```
for(i=0; i<size; i++) //loop through all the letters in this line
```

```
{
```

```
    while(myLineText[j] == ' ') // this is to skip spaces
```

```
    {
```

```
        j++;                //everytime there's a space j gets one more ahead of i
```

```

    }

    mLetterArray[i].ltr=myLineText[j];

    mLetterArray[i].x= j*cxCaps;           //starting x loc of the
letter
    mLetterArray[i].y=(rand() % 200)-200;   //starting y loc of the letter

    mLetterArray[i].rate=rand() % 3;        //random rate for letter 1-3?

    //letter[i].rate=2;

    mLetterArray[i].fadecolor=myLineColor; //assign linecolor passed from
LineInfos
    count++;

    j++;

}

return count;

}

```

```

void SetToTop(COLORREF myColor,DROP mLetterArray[],int myLineLength)

{

```

```

int i;

for(i=0; i<myLineLength; i++)

{

    mLetterArray[i].y=(rand() % 200)-200;

    mLetterArray[i].fadecolor=myColor;

}

}

```

```

void Run(HWND hwnd)

{

    /*-----paint hdc(screen) with the update areas from memdc -----*/

    BitBlt(hdc, 0, 0, cxClient, cyClient, viddc, 0, 0, SRCCOPY);

}

```

```

bool CheckPixel(int X, int Y)

{

    bool yesorno=1;

```

```
    COLORREF colorref;
```

```
    colorref=GetPixel(viddc,X,Y);
```

```
    //0 is closer to dark, 255,255,255 = white
```

```
    if(GetRValue(colorref)<DarkThresh && GetGValue(colorref)<DarkThresh &&  
GetBValue(colorref)<DarkThresh)
```

```
    {
```

```
        yesorno=0;
```

```
    }
```

```
    return yesorno;
```

```
}
```

```
void IncLocAndRgn(DROP myletter[], int mylinelength)
```

```
{
```

```
    int i;
```

```
    int deltadown;
```

```
    int testdown;
```

```

for(i=0; i<mylinelength; i++)
{
    if (myletter[i].y < (480-cyChar)) //letter is still on screen
    {
        //-----INCREMENT y values

        //there is the amount I want to move by, and the amount I want to
test
        //which is a little lower than the move amount

        deltadown=myletter[i].rate + (rand() % 3); //amount we want to
fall
        //testdown = myletter[i].y + deltadown; //where this would end up
        testdown = myletter[i].y + deltadown+ cyChar-1; //where this
would end up + offset

        //test color of pixel you're about to move to

        if(CheckPixel(myletter[i].x,testdown)) //1=white

            //myletter[i].y = testdown; //if ok, set y to this loc

            myletter[i].y += deltadown;

        else

        {

```

while (CheckPixel(myletter[i].x,testdown)==0)//as long as  
checkpix returns 0

testdown-=3;//subtract from testdown until  
checkpixel with this value =1

//deltadown-=3;

myletter[i].y = testdown-cyChar+1;

//myletter[i].y +=deltadown;

}

}

else

//-----OFF BOTTOM -- reset to top

myletter[i].y = 10;

}//end for --gone through whole set of letters

}

//-----Video Callback function

LRESULT CALLBACK VideoCallbackProc(HWND ghWndCap, LPVIDEOHDR lpVHdr)

{

int i;

int LineToStop;

int SecondLineToDrop;

if(!hwnd)

return FALSE;

//-----decompress, mirror and stretch incoming captured frame to vidc

DrawDibDraw(hdd,

backwardsdc,

0,0,

//MM\_TEXT client coordinates, of

the upper left corner of the destination rectangle

(\*pformatbmi).bmiHeader.biWidth,(\*pformatbmi).bmiHeader.biH

eight, //height and width

(LPBITMAPINFOHEADER)pformatbmi,

lpVHdr->lpData,



0,0, //in pixels, of the upper left corner of  
the source rectangle. The coordinates (0,0) represent the upper left corner of the bitmap.

(\*pformatbmi).bmiHeader.biWidth,  
(\*pformatbmi).bmiHeader.biHeight,

DDF\_SAME\_HDC); //DDF\_SAME\_HDC

//----StretchBlt stretches and creates a mirror image of a bitmap--

//If nWidthSrc and nWidthDest have different signs, the function creates a mirror  
image along the x-axis.

//If nHeightSrc and nHeightDest have different signs, the function creates a mirror  
along the y-axis.

StretchBlt( viddc, // handle of destination device context  
640, // x-coordinate of upper-left corner of dest. rect.  
0, // y-coordinate of upper-left corner of dest. rect.  
-640, // width of destination rectangle --nWidthDest  
480, // height of destination rectangle  
backwardsdc, // handle of source device context  
0, // x-coordinate of upper-left corner of source rectangle  
0, // y-coordinate of upper-left corner of source rectangle  
(\*pformatbmi).bmiHeader.biWidth, // width of source  
rectangle --nWidthSrc  
(\*pformatbmi).bmiHeader.biHeight, // height of source  
rectangle

SRCCOPY);

//-----check if line is below a certain level

if(CheckLevel((ActiveLineIndex<gNumLines-1)? ActiveLineIndex+1:0)) //send this  
activelineindex+1,or wrap

{

//-----start active line FADING

LineInfos[ActiveLineIndex].fade=1;

//-----STOP active line -1 from FADING and FALLING (may have to reach back  
to top)

LineToStop=(ActiveLineIndex>0) ? ActiveLineIndex-1: gNumLines-1;

LineInfos[LineToStop].move=0; //stop moving

LineInfos[LineToStop].fade=0; //stop fading

SetToTop(LineInfos[LineToStop].color, Lines[LineToStop].dropline,  
LineLengths[LineToStop]); //reset x and y's for that line to top

//reset line

//-----INCREMENT active line number, and start new line FALLING

```
SecondLineToDrop=(ActiveLineIndex<gNumLines-2)?  
ActiveLineIndex+2:((ActiveLineIndex<gNumLines-1)? 0:1);
```

```
//2
```

```
LineInfos[SecondLineToDrop].move=1; //turn on new line
```

```
ActiveLineIndex=(ActiveLineIndex<gNumLines-1)? ActiveLineIndex+1:0;
```

```
}
```

```
//move, fade and print lines that are both moving and fading
```

```
for (i=0; i<gNumLines; i++)
```

```
{
```

```
    if(LineInfos[i].move && LineInfos[i].fade)
```

```
    {
```

```
        IncLocAndRgn(Lines[i].dropline, LineLengths[i]);
```

```
        //move letters based on vidc image
```

```
        FadeText(Lines[i].dropline, LineLengths[i]);
```

```
        //fade text colors
```

```
        PrintText(Lines[i].dropline, LineLengths[i]);
```

```

    }

}

//move, and print lines that are moving (do this here so these lines are in front of fading
ones)

for (i=0; i<gNumLines; i++)

{

    if(LineInfos[i].move && (!LineInfos[i].fade))//assumes you won't fade w-out
moving.

    {

        IncLocAndRgn(Lines[i].dropline, LineLengths[i]);

        //move letters based on vidc image

        PrintText(Lines[i].dropline, LineLengths[i]);

    }

}

AperatureFix(); //send shadow image to sheild camera

Run(hwnd);//bitblt to scrn

return (LRESULT) TRUE;

}

```

```
void FadeText(DROP myLetter[], int myLineLength)//called once for each line that needs to be faded
```

```
{
```

```
    int i;
```

```
    COLORREF pixelcolor;
```

```
    int scrnR, scrnG, scrnB;
```

```
    int r, g, b;
```

```
    UINT fadedir;
```

```
    int fadeamount=5;
```

```
    //I want things to desaturate as well
```

```
    for (i=0; i<myLineLength; i++)
```

```
    {
```

```
        pixelcolor=GetPixel(viddc, myLetter[i].x + (cxChar/2), myLetter[i].y+(cyChar+1/2));
```

```
        if(myLetter[i].y > 0 && (myLetter[i].fadecolor!=pixelcolor))
```

```
        {
```

```
scrnR=(BYTE)pixelcolor;
```

```
scrnG=(BYTE) (((WORD)pixelcolor) >> 8);
```

```
scrnB=(BYTE) (pixelcolor >> 16);
```

```
r=(BYTE) myLetter[i].fadecolor;
```

```
g=(BYTE) (((WORD)myLetter[i].fadecolor) >> 8);
```

```
b=(BYTE) (myLetter[i].fadecolor >> 16);
```

```
if(r!=scrnR)
```

```
{    //add or subtract fadeamount
```

```
    fadedir=(r>scrnR)? -1:1;
```

```
    r+=fadedir*fadeamount;
```

```
    r=(r>255)? 255:r; //if r>255 make it 255, else let it be
```

```
    r=(r<0)?0:r;
```

```
}
```

```
if(g!=scrnG)
```

```
{    //add or subtract fadeamount
```

```
    fadedir=(g>scrnG)? -1:1;
```

```
    g+=fadedir*fadeamount;
```

```
    g=(g>255)? 255:g; //if r>255 make it 255, else let it be
```

```
    g=(g<0)?0:g;
```

```
}
```

```
if(b!=scrnB)
```

```
{    //add or subtract fadeamount
```

```
    fadedir=(b>scrnB)? -1:1;
```

```
    b+=fadedir*fadeamount;
```

```
    b=(b>255)? 255:b; //if r>255 make it 255, else let it be
```

```
    b=(b<0)?0:b;
```

```
}
```

```
myLetter[i].fadecolor=RGB(r,g,b);
```

```
}
```

```
}//end for
```

```
}
```

```
void PrintText(DROP myLetter[], int myLineLength)
```

```
//called once for each line that needs to be printed
```

```
{
```

```
    int i;
```

```

        for (i=0; i<myLineLength; i++)
        {
            SetTextColor(viddc, myLetter[i].fadecolor);//could do a check here - only set if
necessary
            TextOut(viddc, myLetter[i].x, myLetter[i].y, buffer, wsprintf(buffer,"%c",
myLetter[i].ltr));
        }
    }

```

```

bool CheckLevel(int myLine)

```

```

{
    int i=0;

    int numBelowLevel=0;

    while(i<LineLengths[myLine])
    {
        //loop through letter array for this line, increment belowLevel for all letters below
100 y coord
        if(Lines[myLine].dropline[i].y > 100)
            numBelowLevel++;

        i++;
    }

    if(numBelowLevel> (LineLengths[myLine]/2))

```



```
return 1;
```

```
else
```

```
return 0;
```

```
}
```

```
//-----
```

```
int FindDriver(char driverstring[80], int stringlength)
```

```
{
```

```
    char szDeviceName[80];           //string for camera names
```

```
    char szDeviceVersion[80];       //string for camera versions
```

```
    int wIndex;                      //camera driver index numbers
```

```
    int j;                          //for looping through driver string
```

```
    bool match=1;
```

```
//-----test which video drivers are available
```



```
}
```

```
void CleanUp()
```

```
{
```

```
//-----disconnect driver
```

```
capDriverDisconnect(ghWndCap);
```

```
//-----clean up GDI stuff
```

```
DeleteDC(viddc);
```

```
DeleteDC(hdc);
```

```
DeleteObject(tempbitmap2);
```

```
DeleteDC(backwardsdc);
```

```
DeleteObject(BackwardsBitmap);
```

```
//-----close DrawDibLib
```

```
DrawDibClose(hdd);
```

```
//-----be sure cursor value is set back to 0
```

```
if(CursorValue!=0)
```

```
{
```

```
while(CursorValue<0)
```

```
CursorValue=ShowCursor(1);
```

```
while(CursorValue>0)
```

```
CursorValue=ShowCursor(0);
```

```
}
```

```
}
```

```
void AperatureTest(){
```

```
HBRUSH oldbrush;
```

```
HPEN oldpen;
```

```
oldpen=SelectObject(hdc,CreatePen(PS_SOLID,1,RGB(HoleGrey,HoleGrey,HoleGrey)));
//selects new brush into hdc
```

```
oldbrush=SelectObject(hdc,CreateSolidBrush(RGB(HoleGrey,HoleGrey,HoleGrey)));
//CONST LOGBRUSH *lplb
```

```
//RGB(HoleGrey,HoleGrey,HoleGrey)
```

```
Ellipse(    hdc, // handle to device context
           HoleL, // x-coord. of bounding rectangle's upper-left corner
           HoleT, // y-coord. of bounding rectangle's upper-left corner
           HoleL+HoleWidth, // x-coord. of bounding rectangle's lower-right
           corner
           HoleT+HoleHeight // y-coord. bounding rectangle's f lower-right
           corner
           );
```

```
DeleteObject(SelectObject(hdc,oldpen)); //selects oldbrush and deletes one we created.
```

```
DeleteObject(SelectObject(hdc,oldbrush));
```

```
//The ellipse is outlined by using the current pen and is filled by using the current brush.
```

```
}
```

```
void AperatureFix(){
```

```
    HBRUSH oldbrush;
```

```
    HPEN oldpen;
```

```
    oldpen=SelectObject(viddc,CreatePen(PS_SOLID,1,RGB(HoleGrey,HoleGrey,HoleGrey)));  
    //selects new brush into hdc
```

```
        oldbrush=SelectObject(viddc,CreateSolidBrush(RGB(HoleGrey,HoleGrey,HoleGrey)));  
    //CONST LOGBRUSH *lplb
```

```
        //RGB(HoleGrey,HoleGrey,HoleGrey)
```

```
    Ellipse(        viddc, // handle to device context
```

```
        HoleL, // x-coord. of bounding rectangle's upper-left corner
```

```
        HoleT, // y-coord. of bounding rectangle's upper-left corner
```

```
        HoleL+HoleWidth, // x-coord. of bounding rectangle's lower-right  
corner
```



```

Rectangle(    whichdc, // handle of device context

             left, // x-coord. of bounding rectangle's upper-left corner

             top, // y-coord. of bounding rectangle's upper-left corner

             right, // x-coord. of bounding rectangle's lower-right corner

             bottom // y-coord. of bounding rectangle's lower-right corner

            );

```

```

DeleteObject(SelectObject(viddc,oldpen)); //selects oldbrush and deletes one we created.

```

```

DeleteObject(SelectObject(viddc,oldbrush));

```

```

}

```

```

bool ReadInTextFromFile(LPCTSTR szFileName,LPVOID szBuffer,DWORD
maxcharstoread,LPDWORD lpcharsread){

```

```

HANDLE myHandle;          //returned by CreateFile

```



```
bool bReadFileOK;    //to test createfile (though also using formatMessage to do this)
```

```
bool myResult=0;      //returned by ReadFile
```

```
myHandle=CreateFile(szFileName,
```

```
                        GENERIC_READ,
```

```
                        FILE_SHARE_READ,
```

```
                        NULL,
```

```
                        OPEN_EXISTING,
```

```
                        FILE_ATTRIBUTE_NORMAL,
```

```
                        NULL);
```

```
if(myHandle==INVALID_HANDLE_VALUE){
```

```
    //----CreateFile returned an INVALID FILE HANDLE
```

```
    //TextOut(hdc, 400, 10, buffer, wsprintf(buffer, "error in createfile"));
```

```
    //--set our tracking var
```

```
    bReadFileOK=FALSE;
```

```
}
```

```
else{
```

```
//----CreateFile returned an VALID FILE HANDLE
```

```
//--read the file
```

```
myResult = ReadFile(myHandle, // handle of file to read
```

```
szBuffer, // address of buffer that  
receives data LPVOID
```

```
maxcharstoread, // number of bytes  
to read nNumberOfBytesToRead MAXREAD
```

```
lpcharsread, // address of number of  
bytes read
```

```
NULL); // address of structure for  
overlapped data
```

```
if (myResult!=0){
```

```
//--ReadFile PROCESSED OK
```

```
bReadFileOK=TRUE;
```

```
//TextOut(hdc, 10, 50, buffer, wsprintf(buffer,  
"NumberOfBytesRead = %d", NumberOfBytesRead ));
```

```
//TextOut(hdc, 10, 130, buffer, wsprintf(buffer, "textlength= %d",  
strlen(szBuffer)));
```

be closed

CloseHandle(myHandle); //readfile was successful so this needs to

}else{

//--ReadFile DID NOT PROCESS OK

bReadFileOK=FALSE;

}

}//end else for createfile

return bReadFileOK;

}